MBA em Ciência de Dados

Análise de Dados com Base em Processamento Massivo em Paralelo

Avaliação Final

Material Produzido por:

> **Profa. Dra. Cristina Dutra de Aguiar**

CEMEAI - ICMC/USP São Carlos

As questões desta avaliação final estão espalhadas ao longo do texto. Por favor, procurem por Questão para encontrar a especificação das questões e o local no qual cada questão deve ser respondida. Também é possível localizar as questões utilizando o menu de navegação. O *notebook* contém a constelação de fatos da BI Solutions que deve ser utilizada para responder às questões e também todas as bibliotecas, bases de dados, inicializações, instalações, importações, geração de dataFrames, geração de visões temporárias e conversão dos tipos de dados necessárias para a realização da questão. Portanto, o notebook está preparado para ser executado usando Pandas, o método spark.sql() e os métodos do módulo pyspark.sql.

O uso do *framework* Spark requer diversas configurações no ambiente de desenvolvimento para executar o *notebook*. Dado que tal complexidade foge do escopo de nossa disciplina, recomenda-se que o *notebook* seja executado na plataforma de desenvolvimento COLAB. O uso do COLAB proporciona um ambiente de desenvolvimento pré-configurado e remove a complexidade de instalação e configuração de pacotes e *frameworks* que são utilizados na disciplina.

IMPORTANTE

Antes de fazer a avaliação, leia atentamente a seção 5, que detalha instruções importantes sobre a avaliação e o critério de correção.

INSTRUÇÕES DE ENTREGA

O que deve ser entregue:

- **O notebook com as respostas no formato .ipynb**
- **O notebook com as respostas no formato .pdf**

Ambos arquivos devem ser nomeados usando o primeiro nome e o último sobrenome do aluno. Por exemplo: CristinaAguiar.ipynb e CristinaAguiar.pdf.

Boa avaliação!

1 Constelação de Fatos da BI Solutions

A aplicação de *data warehousing* da BI Solutions utiliza como base uma constelação de fatos, conforme descrita a seguir.

Tabelas de dimensão

- data (dataPK, dataCompleta, dataDia, dataMes, dataBimestre, dataTrimestre, dataSemestre, dataAno)
- funcionario (funcPK, funcMatricula, funcNome, funcSexo, funcDataNascimento, funcDiaNascimento, funcMesNascimento, funcAnoNascimento, funcCidade, funcEstadoNome, funcEstadoSigla, funcRegiaoNome, funcRegiaoSigla, funcPaisNome, funcPaisSigla)
- equipe (equipePK, equipeNome, filialNome, filialCidade, filialEstadoNome, filialEstadoSigla, filialRegiaoNome, filialRegiaoSigla, filialPaisNome, filialPaisSigla)
- cargo (cargoPK, cargoNome, cargoRegimeTrabalho, cargoEscolaridadeMinima, cargoNivel)
- cliente (clientePK, clienteNomeFantasia, clienteSetor, clienteCidade, clienteEstadoNome, clienteEstadoSigla, clienteRegiaoNome, clienteRegiaoSigla, clientePaisNome, clientePaisSigla)

Tabelas de fatos

- pagamento (dataPK, funcPK, equipePK, cargoPK, salario, quantidadeLancamentos)
- negociacao (dataPK, equipePK, clientePK, receita, quantidadeNegociacoes)

2 Obtenção dos Dados da BI Solutions

2.1 Baixando o Módulo wget

Para baixar os dados referentes ao esquema relacional da constelação de fatos da BI Solutions, é utilizado o módulo **wget**. O comando a seguir realiza a instalação desse módulo.

```
#instalando o módulo wget
%%capture
!pip install -q wget
!mkdir data
```

2.2 Obtenção dos Dados das Tabelas de Dimensão

Os comandos a seguir baixam os dados que povoam as tabelas de dimensão.

```
#baixando os dados das tabelas de dimensão
import wget

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/data.csv"
wget.download(url, "data/data.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/funcionar
wget.download(url, "data/funcionario.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/equipe.cs
wget.download(url, "data/equipe.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/cargo.csv
wget.download(url, "data/cargo.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/cliente.c
wget.download(url, "data/cliente.csv")
```

```
'data/cliente.csv'
```

2.3 Obtenção dos Dados Tabelas de Fatos

Os comandos a seguir baixam os dados que povoam as tabelas de fatos.

```
#baixando os dados das tabelas de fatos
url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/pagamento
wget.download(url, "data/pagamento.csv")

url = "https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solutions/main/negociaca
wget.download(url, "data/negociacao.csv")
```

```
'data/negociacao.csv'
```

3 Apache Spark Cluster

3.1 Instalação

Neste *notebook* é criado um *cluster* Spark composto apenas por um **nó mestre**. Ou seja, o *cluster* não possui um ou mais **nós de trabalho** e o **gerenciador de cluster**. Nessa configuração, as tarefas (*tasks*) são realizadas no próprio *driver* localizado no **nó mestre**.

Para que o cluster possa ser criado, primeiramente é instalado o Java Runtime Environment (JRE) versão 8.

```
#instalando Java Runtime Environment (JRE) versão 8
```

```
%%capture
!apt-get remove openjdk*
!apt-get update --fix-missing
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Na sequência, é feito o *download* do Apache Spark versão 3.0.0.

```
#baixando Apache Spark versão 3.0.0
%%capture
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.7.tgz
!tar xf spark-3.0.0-bin-hadoop2.7.tgz && rm spark-3.0.0-bin-hadoop2.7.tgz
```

Na sequência, são configuradas as variáveis de ambiente JAVA_HOME e SPARK_HOME. Isto permite que tanto o Java quanto o Spark possam ser encontrados.

```
import os
#configurando a variável de ambiente JAVA_HOME
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
#configurando a variável de ambiente SPARK_HOME
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop2.7"
```

Por fim, são instalados dois pacotes da linguagem de programação Python, cujas funcionalidades são descritas a seguir.

> **Pacote findspark:** Usado para ler a variável de ambiente SPARK_HOME e armazenar seu valor na variável dinâmica de ambiente PYTHONPATH. Como resultado, Python pode encontrar a instalação do Spark.

> **Pacote pyspark:** PySpark é a API do Python para Spark. Ela possibilita o uso de Python, considerando que o *framework* Apache Spark encontra-se desenvolvido na linguagem de programação Scala.

```
%%capture
#instalando o pacote findspark
!pip install -q findspark==1.4.2
#instalando o pacote pyspark
!pip install -q pyspark==3.0.0
```

3.2 Conexão

PySpark não é adicionado ao *sys.path* por padrão. Isso significa que não é possível importá-lo, pois o interpretador da linguagem Python não sabe onde encontrá-lo.

Para resolver esse aspecto, é necessário instalar o módulo `findspark`. Esse módulo mostra onde PySpark está localizado. Os comandos a seguir têm essa finalidade.

```
#importando o módulo findspark
import findspark
#carregando a variávels SPARK_HOME na variável dinâmica PYTHONPATH
findspark.init()
```

Depois de configurados os pacotes e módulos e inicializadas as variáveis de ambiente, é possível iniciar o uso do Spark na aplicação de `data warehousing`. Para tanto, é necessário importar o comando `SparkSession` do módulo `pyspark.sql`. São utilizados os seguintes conceitos:

- `SparkSession`: permite a criação de `DataFrames`. Como resultado, as tabelas relacionais podem ser manipuladas por meio de `DataFrames` e é possível realizar consultas OLAP por meio de comandos SQL.
- `builder`: cria uma instância de SparkSession.
- `appName`: define um nome para a aplicação, o qual pode ser visto na interface de usuário web do Spark.
- `master`: define onde está o nó mestre do *cluster*. Como a aplicação é executada localmente e não em um *cluster*, indica-se isso pela *string* `local` seguida do parâmetro `[*]`. Ou seja, define-se que apenas núcleos locais são utilizados.
- `getOrCreate`: cria uma SparkSession. Caso ela já exista, retorna a instância existente.

Observação: A lista completa de todos os parâmetros que podem ser utilizados na inicialização do *cluster* pode ser encontrada neste link.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("pyspark-notebook").master("local[*]").getOrCreate()
```

4 Geração dos DataFrames em Pandas da BI Solutions

Nesta seção são gerados os DataFrames em Pandas. Atenção aos nomes desses DataFrames.

```
import pandas as pd
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

```
cargoPandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solu
clientePandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_So
dataPandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Solut
equipePandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_Sol
funcionarioPandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_E
negociacaoPandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI
pagamentoPandas = pd.read_csv('https://raw.githubusercontent.com/GuiMuzziUSP/Data_Mart_BI_
```

5 Geração dos DataFrames em Spark da BI Solutions

Nesta seção são gerados dos DataFrames em Spark. Atenção aos nomes desses DataFrames.

5.1 Criação dos DataFrames

```
#criando os DataFrames em Spark
cargo = spark.read.csv(path="data/cargo.csv", header=True, sep=",")
cliente = spark.read.csv(path="data/cliente.csv", header=True, sep=",")
data = spark.read.csv(path="data/data.csv", header=True, sep=",")
equipe = spark.read.csv(path="data/equipe.csv", header=True, sep=",")
funcionario = spark.read.csv(path="data/funcionario.csv", header=True, sep=",")
negociacao = spark.read.csv(path="data/negociacao.csv", header=True, sep=",")
pagamento = spark.read.csv(path="data/pagamento.csv", header=True, sep=",")
```

5.2 Atualização dos Tipos de Dados

Nos comandos a seguir, primeiro são identificados quais colunas de quais `DataFrames` devem ser do tipo de dado inteiro. Na sequência, ocorre a conversão. Por fim, são exibidos os esquemas dos `DataFrames`, possibilitando visualizar a mudança de tipo de dados das colunas especificadas.

```
# identificando quais colunas de quais DataFrames devem ser do tipo de dado inteiro
colunas_cargo = ["cargoPK"]
colunas_cliente = ["clientePK"]
colunas_data = ["dataPK", "dataDia", "dataMes", "dataBimestre", "dataTrimestre", "dataSeme
colunas_equipe = ["equipePK"]
colunas_funcionario = ["funcPK", "funcDiaNascimento", "funcMesNascimento", "funcAnoNascime
colunas_negociacao = ["equipePK", "clientePK", "dataPK", "quantidadeNegociacoes"]
colunas_pagamento = ["funcPK", "equipePK", "dataPK", "cargoPK", "quantidadeLancamentos"]
```

```
# importando o tipo de dado desejado
from pyspark.sql.types import IntegerType
```

```
# atualizando o tipo de dado das colunas especificadas
# substituindo as colunas já existentes

for coluna in colunas_cargo:
  cargo = cargo.withColumn(coluna, cargo[coluna].cast(IntegerType()))

for coluna in colunas_cliente:
  cliente = cliente.withColumn(coluna, cliente[coluna].cast(IntegerType()))
```

```
for coluna in colunas_data:
  data = data.withColumn(coluna, data[coluna].cast(IntegerType()))

for coluna in colunas_equipe:
  equipe = equipe.withColumn(coluna, equipe[coluna].cast(IntegerType()))

for coluna in colunas_funcionario:
  funcionario = funcionario.withColumn(coluna, funcionario[coluna].cast(IntegerType()))

for coluna in colunas_negociacao:
  negociacao = negociacao.withColumn(coluna, negociacao[coluna].cast(IntegerType()))

for coluna in colunas_pagamento:
  pagamento = pagamento.withColumn(coluna, pagamento[coluna].cast(IntegerType()))
```

Nos comandos a seguir, primeiro são identificados quais colunas de quais `DataFrames` devem ser do tipo de dado número de ponto flutuante. Na sequência, ocorre a conversão. Por fim, são exibidos os esquemas dos `DataFrames`, possibilitando visualizar a mudança de tipo de dados das colunas especificadas.

```
# identificando quais colunas de quais DataFrames devem ser do tipo de dado número de pont
colunas_negociacao = ["receita"]
colunas_pagamento = ["salario"]


# importando o tipo de dado desejado
from pyspark.sql.types import FloatType


# atualizando o tipo de dado das colunas especificadas
# substituindo as colunas já existentes

for coluna in colunas_negociacao:
  negociacao = negociacao.withColumn(coluna, negociacao[coluna].cast(FloatType()))

for coluna in colunas_pagamento:
  pagamento = pagamento.withColumn(coluna, pagamento[coluna].cast(FloatType()))


# importando funções adicionais
from pyspark.sql.functions import round, desc
```

5.3 Criação de Visões Temporárias

```
#criando as visões temporárias
cargo.createOrReplaceTempView("cargo")
cliente.createOrReplaceTempView("cliente")
data.createOrReplaceTempView("data")
equipe.createOrReplaceTempView("equipe")
funcionario.createOrReplaceTempView("funcionario")
```

```
negociacao.createOrReplaceTempView("negociacao")
pagamento.createOrReplaceTempView("pagamento")
```

6 Instruções Importantes sobre a Avaliação

6.1 Especificação das Consultas OLAP

As consultas OLAP devem ser respondidas de acordo com o solicitado em cada questão. As seguintes solicitações podem ser feitas:

- Resolva a questão especificando a consulta OLAP usando **Pandas**. Neste caso, a consulta deve ser respondida usando os conceitos apresentados na Aula 05 da disciplina. Ou seja, a consulta deve ser respondida usando os métodos disponíveis na biblioteca Pandas para uso em Python. Não é possível usar o método `spark.sql()` para especificar a consulta. Também não é possível usar os demais métodos do módulo `pyspark.sql` para especificar a consulta.
- Resolva a questão especificando a consulta OLAP na **linguagem SQL**. Neste caso, a consulta deve ser respondida usando os conceitos apresentados na Aula 07 da disciplina. Ou seja, a consulta deve ser respondida usando a linguagem SQL textual e o método `spark.sql()`. Não é possível usar os métodos disponíveis na biblioteca Pandas para especificar a consulta. Também não é possível usar os demais métodos do módulo `pyspark.sql` para especificar a consulta, com exceção do método `show()` para listar o resultado da consulta.
- Resolva a questão especificando a consulta OLAP usando os **métodos de pyspark.sql**. Neste caso, a consulta deve ser respondida usando os conceitos apresentados na Aula 08 da disciplina. Ou seja, a consulta deve ser respondida usando os demais métodos do módulo `pyspark.sql`. Não é possível usar os métodos disponíveis na biblioteca Pandas para especificar a consulta. Também não é possível usar o método `spark.sql()` para especificar a consulta.

AVISO: Caso a consulta seja especificada de forma diferente do que foi solicitado, a resposta não será considerada, mesmo que ela esteja correta.

6.2 Ordem das Colunas e das Linhas

A resolução das questões deve seguir estritamente as especificações definidas em cada consulta. Isto significa que:

- As **colunas** solicitadas devem ser exibidas exatamente na mesma ordem que a definida na questão. Note que todas as colunas a serem exibidas como resposta da consulta, bem como a ordem na qual elas devem aparecer são sempre definidas na questão.

- As **linhas** retornadas como respostas devem ser exibidas exatamente na mesma ordem que a definida na questão. Note que a ordem na qual as linhas devem aparecer são sempre definidas na questão.
- Os **nomes das colunas** renomeadas devem seguir estritamente os nomes definidos na questão. Para evitar possíveis erros, os nomes das colunas renomeadas não possuem acentos e espaços em branco, além de serem escritos utilizando apenas letras maiúsculas. Note que os nomes das colunas renomeadas são sempre definidos na questão.

AVISO: Essas orientações devem ser seguidas uma vez que a correção da avaliação será realizada de forma automática. Caso a consulta retorne resultados de forma diferente do que foi solicitado, a resposta não será considerada, mesmo que ela esteja correta.

6.3 Listagem das Respostas das Consultas

A resposta de cada consulta deve ser listada usando o método `show()`. Nenhum outro método pode ser utilizado com essa finalidade.

Devem ser listadas apenas as `25` primeiras linhas de resposta de cada consulta. Adicionalmente, devem ser listadas *strings* com tamanho maior do que 20 caracteres, ou seja, o parâmetro `truncate` do método `show()` deve ser inicializado como `false`.

Portanto, a listagem das respostas deve ser feita utilizando o método `show()` como especificado a seguir.

- Quando a consulta OLAP for especificada usando **Pandas**. Utilize o comando `df.head(25)` para exibir o resultado da consulta.
- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize o comando `spark.sql(consultaSQL).show(25,truncate=False)` para exibir o resultado da consulta.
- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize o comando `nomeDoDataFrame.show(25,truncate=False)` para exibir o resultado da consulta.

6.4 Arredondamento dos Dados

Deve ser realizado o arredondamento dos dados todas as vezes que uma função de agregação for aplicada às medidas numéricas `salario` da tabela de dimensão `pagamento` e `receita` da tabela de dimensão `negociacao`.

O arredondamento deve ser realizado usando a função `round()` na linguagem SQL e o método `round()` em `pyspark.sql` e deve arredondar os dados até duas casas decimais. Por exemplo, podem ser produzidos resultados da forma `112233.4` e `112233.44`.

Portanto, o arredondamento dos dados deve ser feito como especificado a seguir.

- Quando a consulta OLAP for especificada usando **Pandas**. Utilize o comando `df.round(2)` para arredondar os dados até duas casas decimais.
- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize a função `ROUND(funçãoDeAgregação,2)` para arredondar o dado até duas casas decimais.
- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize o método `round(funçãoDeAgregação,2)` para arredondar o dado até duas casas decimais.

6.5 Respostas das Questões

As respostas das questões devem ser fornecidas de duas formas diferentes:

- Exibidas na saída padrão.
- Armazenadas em um arquivo no formato csv.

A seguir são detalhadas as instruções em **Pandas**, **spark.sql()** e **pyspark()** para que as respostas sejam mostradas de forma apropriada para as correções.

Em **Pandas**

```
# Resolve a questão usando a variável de nome questao
QuestaoX = consulta.round(2).head(25)
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Exibe a resposta da questão na saída padrão
display(QuestaoX)
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Gera o arquivo no formato csv com a resposta da questão
QuestaoX.to_csv("questaoX.csv", index=False, header=True)
onde X é o número da questão, por exemplo Questao1, Questao2, ...
```

Em spark.sql()

```
# Resolve a questão usando a variável de nome questao
QuestaoX = spark.sql(query)
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Exibe a resposta da questão na saída padrão
QuestaoX.show(25, truncate=False)
```

```
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Gera o arquivo no formato csv com a resposta da questão
QuestaoX\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("questaoX.csv", index=False, header=True)
onde X é o número da questão, por exemplo Questao1, Questao2, ...
```

Em pyspark()

```
# Resolve a questão usando a variável de nome questao
QuestaoX = consulta
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Exibe a resposta da questão na saída padrão
QuestaoX.show(25, truncate=False)
onde X é o número da questão, por exemplo Questao1, Questao2, ...

# Gera o arquivo no formato csv com a resposta da questão
QuestaoX\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("QuestaoX.csv", index=False, header=True)
onde X é o número da questão, por exemplo Questao1, Questao2, ...
```

6.6 Comentários Explicativos

Devem ser colocados comentários no código que expliquem o passo a passo da resolução da questão. Os comentários explicativos devem ser realizados como especificado a seguir.

- Quando a consulta OLAP for especificada usando **Pandas**. Utilize `#` para colocar comentário. Por exemplo:

```
# para a solução desta consulta OLAP, primeiramente é aplicado o método ... para .....
# Na sequência, é aplicado o método ... para ...
```

- Quando a consulta OLAP for especificada usando a **linguagem SQL**. Utilize `#` para colocar comentários gerais (conforme explicado para os demais métodos de `pyspark.sql`) ou utilize `--` para colocar comentários no comando SQL. Por exemplo:

```
# neste comentário são descritas as características de cada cláusula da consulta SQL.
# A funcionalidade da cláusula SELECT nesta consulta é ...
```

```
# A funcionalidade da cláusula FROM nesta consulta é ...
# A funcionalidade da cláusula WHERE nesta consulta é ...


-- A funcionalidade da cláusula SELECT nesta consulta é ...
SELECT funcNome
-- A funcionalidade da cláusula FROM nesta consulta é ...
FROM funcionario
-- A funcionalidade da cláusula WHERE nesta consulta é ...
WHERE funcPK = 1
```

- Quando a consulta OLAP for especificada usando os demais **métodos de pyspark.sql**. Utilize `#` para colocar comentário. Por exemplo:

```
# para a solução desta consulta OLAP, primeiramente é aplicado o método ... para .....
# Na sequência, é aplicado o método ... para ...
```

6.7 Indentação e Organização

As consultas e os comandos que respondem às questões dessa avaliação devem ser escritos de forma indentada. Em caso de dúvida, observem os *notebooks* da Aula 05, da Aula 07 e da Aula 08 e verifiquem como as consultas e os comandos foram indentados.

AVISO: Com relação à organização, é necessário que as respostas às questões sejam localizadas aonde especificado no *notebook*. Por favor, procurem por "Resposta da Questão" para encontrar o local no qual as respostas devem ser especificadas. Também é possível localizar o local das respostas utilizando o menu de navegação.

6.8 Critério de Avaliação

Na correção da avaliação, serão ponderados os seguintes aspectos:

- Corretude da execução das consultas OLAP.
- Atendimento às especificações definidas nas seções 6.1, 6.2, 6.3, 6.4, 6.5.
- Atendimento às especificações da sintaxe das cláusulas e dos métodos utilizados para resolver cada questão.
- Qualidade da documentação entregue, de acordo com as especificações definidas nas seções 6.6 e 6.7.

7 Consultas OLAP

O objetivo das consultas OLAP é realizar diferentes investigações sobre aspectos específicos no que tange às atividades realizadas pela BI Solutions. Os resultados obtidos nas investigações poderão ser posteriormente utilizados para a definição de estratégias que a empresa deve executar para prover melhorias.

7.1 Análises Relacionadas aos Cargos

Foi identificado que, nos últimos anos, o cargo de nome "ADMINISTRADOR EM SEGURANCA DA INFORMACAO" teve um aumento expressivo no que tange aos gastos com salários. O objetivo das análises desta seção é obter uma visão relacionada a esse aspecto, por meio da investigação dos gastos em salários considerando diferentes fatores.

Podem ser realizadas diferentes análises, sendo que três delas são solicitadas a seguir.

Questão 1 (valor 1,0)

Liste, para cada ano, a soma dos salários para o cargo de nome "ADMINISTRADOR EM SEGURANCA DA INFORMACAO". Arredonde a soma dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ANO", "TOTALDESPESA". Ordene as linhas exibidas primeiro pelo total de despesa em ordem descendente e depois pelo ano em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP usando Pandas.

Clique duas vezes (ou pressione "Enter") para editar

Resposta da Questão 1

```
# Resposta da Questão 1

# Fazendo a consulta especificada
# Primeiro fazer o merge de pagamentoPandas com dataPandas e especificar as chaves primári
# Em seguida filtrar o cargoNome desejado com query() e aplicar novo merge()
# Depois renomear as colunas com rename(), agrupar com groupby() e transformar em datafram
# Enfim, ordenar com sort_values() e ascending()

pagamentoData = pagamentoPandas.merge(dataPandas, on = 'dataPK')
cargoFiltrado = cargoPandas.query('cargoNome == "ADMINISTRADOR EM SEGURANCA DA INFORMACAO'
pagamentoDataCargoFiltrado = pagamentoData.merge(cargoFiltrado, on = 'cargoPK')
Questao1 = pagamentoDataCargoFiltrado.rename(mapper={"dataAno": "ANO", "salario": "TOTALDE
            .groupby(['ANO'])['TOTALDESPESA'].sum().to_frame()
Questao1 = Questao1.sort_values(by=['TOTALDESPESA', 'ANO'], ascending=[False, False])
```

```
# arredondando as casas decimais e finalizando a consulta mostrando seu resultado com .hea
Questao1 = Questao1.round(2).head(25)

# exibindo a resposta na saída padrão
display(Questao1)

# gerando o arquivo no formato csv com a resposta da questão
Questao1.to_csv("Questao1.csv", index=False, header=True)
```

ANO	TOTALDESPESA
2020	1883273.28
2019	1883273.28
2018	1239394.80
2017	943759.32
2016	475625.52

Questão 2 (valor: 1,0)

Liste, para cada nome da região da filial, a soma dos salários para o cargo de nome "ADMINISTRADOR EM SEGURANCA DA INFORMACAO". Arredonde a soma dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "REGIAO", "TOTALDESPESA". Ordene as linhas exibidas primeiro pelo total de despesa em ordem descendente e depois pelo nome da região em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

Resposta da Questão 2

```
# Resposta da Questão 2

# Fazendo a consulta especificada
# Primeiro selecionar as colunas de saída com SELECT...
# ...agregando salario e despesa com SUM(), arredondando com ROUND() e renomeando com AS `
# Depois fazer o JOIN das tabelas necessárias especificando as chaves primárias
# Na sequência filtrar com WHERE o cargoNome desejado
# Enfim, agrupar com GROUP BY e ordenar a saída na sequência desejada com ORDER BY DESC

query = """
SELECT filialRegiaoNome AS `REGIAO`, ROUND(SUM(salario),2) AS `TOTALDESPESA`
FROM pagamento JOIN equipe ON pagamento.equipePK = equipe.equipePK
               JOIN cargo ON pagamento.cargoPK = cargo.cargoPK
WHERE cargoNome = 'ADMINISTRADOR EM SEGURANCA DA INFORMACAO'
GROUP BY REGIAO
```

```
ORDER BY TOTALDESPESA DESC, REGIAO DESC
"""

# Guardando a resposta no nome da questão
Questao2 = spark.sql(query)

# Exibindo a resposta da questão na saída padrão com .show()
Questao2.show(25, truncate=False)

## Gerando o arquivo no formato csv com a resposta da questão
Questao2\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("Questao2.csv", index=False, header=True)
```

```
+------------+------------+
|REGIAO      |TOTALDESPESA|
+------------+------------+
|SUDESTE     |3923904.22  |
|CENTRO-OESTE|1569126.83  |
|NORDESTE    |932295.13   |
+------------+------------+
```

Questão 3 (valor: 1,0)

Liste, por sexo, a soma dos salários para o cargo de nome "ADMINISTRADOR EM SEGURANCA DA INFORMACAO". Arredonde a soma dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "SEXO", "TOTALDESPESA". Ordene as linhas exibidas primeiro pelo total de despesa em ordem descendente e depois pelo sexo em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP usando os métodos de pyspark.sql

Resposta da Questão 3

```
# Resposta da Questão 3

# Resolvendo a questão
# Primeiro fazer o join() das tabelas necessárias especificando as chaves primárias com or
# Filtrar o cargoNome com where()
# Selecionar as colunas de saída com select()
# Agrupar por sexo com groupBy() e agregar salario com sum()
# Arredondar os valores com withColumn() e round()
# Renomear as colunas com withColumnRenamed
# Ordenar com orderBy(desc())

Questao3 = pagamento\
   .join(funcionario, on="funcPK")\
   .join(cargo, on="cargoPK")\
```

```
    .where("cargoNome = 'ADMINISTRADOR EM SEGURANCA DA INFORMACAO'")\
    .select("funcSexo", "salario")\
    .groupBy("funcSexo")\
    .sum("salario")\
    .withColumn("sum(salario)", round("sum(salario)",2))\
    .withColumnRenamed("sum(salario)", "TOTALDESPESA")\
    .withColumnRenamed("funcSexo", "SEXO")\
    .orderBy(desc("TOTALDESPESA"))

# Exibindo a resposta da questão na saída padrão com .show()
Questao3.show(25, truncate=False)

# # Gerando o arquivo no formato csv com a resposta da questão
Questao3\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("Questao3.csv", index=False, header=True)
```

```
+----+------------+
|SEXO|TOTALDESPESA|
+----+------------+
|M   |4380094.08  |
|F   |2045232.1   |
+----+------------+
```

7.2 Análises Relacionadas às Regiões

Foi identificada a necessidade de se investigar despesas e receitas no que tange às regiões. O objetivo das análises desta seção é obter uma visão relacionada a esse aspecto.

Podem ser realizadas diferentes análises, sendo que três delas são solicitadas a seguir.

Questão 4 (valor: 1,0)

Liste, para cada nome do estado da filial, a soma das receitas por ano considerando apenas o trimestre 1 e os clientes cuja região na qual eles moram é a mesma região na qual a filial está localizada. Arredonde a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ESTADO", "ANO", "TOTALRECEITA". Ordene as linhas exibidas primeiro pelo total de receitas em ordem descendente, depois por estado em ordem descendente, depois pelo ano em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP usando Pandas.

Resposta da Questão 4

```
# Resposta da Questão 4
# Fazendo a consulta especificada
```

```
# Primeiro filtrar o trimestre desejado com query()
# Em seguida, fazer o merge de negociacaoPandas com a data filtrada
# Depois, novo merge com clientePandas e equipePandas especificando as chaves primárias cc
# Filtrar onde 'filialRegiaoNome == clienteRegiaoNome' com query()
# Renomear as colunas com rename()
# Agrupar Estado e Ano com groupby(), agregando a receita com sum() e transformando em dat
# Ordenar com sort_values() e ascending

dataFiltrada = dataPandas.query('dataTrimestre == 1')
receitaData = negociacaoPandas.merge(dataFiltrada, on = 'dataPK')
receitaDataCliente = receitaData.merge(clientePandas, on = 'clientePK')
receitaDataClienteEquipe = receitaDataCliente.merge(equipePandas, on = 'equipePK')
receitaDataClienteEquipeFiltrada = receitaDataClienteEquipe.query('filialRegiaoNome == cli
Questao4 = \
receitaDataClienteEquipeFiltrada\
.rename(mapper={"filialEstadoNome": "ESTADO", "dataAno": "ANO", "receita": "TOTALRECEITA"}
.groupby(['ESTADO', 'ANO'])['TOTALRECEITA'].sum().to_frame()
Questao4 =Questao4.sort_values(by=['TOTALRECEITA', 'ESTADO', 'ANO'], ascending=[False, Fal

# arredondando as casas decimais e finalizando a consulta mostrando seu resultado com .hea
Questao4 = Questao4.round(2).head(25)

# exibindo a resposta na saída padrão
display(Questao4)

# gerando o arquivo no formato csv com a resposta da questão
Questao4.to_csv("Questao4.csv", index=False, header=True)
```

		TOTALRECEITA
SAO PAULO	**2017**	313013.85
SAO PAULO	**2016**	241678.25

Questão 5 (valor: 1,0)

Liste, para cada nome da região da filial, a soma dos salários e a soma das receitas, considerando apenas o ano de 2017. Arredonde a soma dos salários e a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "REGIAO", "TOTALRECEITAEQUIPE", "TOTALDESPESAEQUIPE". Ordene as linhas exibidas primeiro pelo total de receitas em ordem descendente, depois pelo total de despesas ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

Resposta da Questão 5

```
# Resposta da Questão 5

'''
COMENTÁRIO/EXPLICAÇÃO

A query abaixo foi construída no modelo de operação "Drill-Across", em que as
duas tabelas fato ("pagamento" e "negociacao") foram consultadas, filtradas e
agrupadas individualmente e o resultado dessas duas operações foi juntado no
resultado final (ou seja, fazendo a junção pela região).

Para a fato "pagamento", juntamos as dimensões "data" e "equipe" a partir de
suas respectivas chaves. Filtramos o ano de 2017, agrupamos por região da filial
e calculamos a soma das despesas (salários).

O mesmo processo foi feito para a fato "negociacao", alterando apenas a variável
agregada para a receita (não os salários).

Feitas as duas "subconsultas", juntamos ambas pela região e montamos a saída
conforme solicitado.
'''

query = '''
SELECT despesas.REGIAO, TOTALRECEITAEQUIPE, TOTALDESPESAEQUIPE
FROM (
  SELECT filialRegiaoNome AS REGIAO, ROUND(SUM(salario), 2) AS TOTALDESPESAEQUIPE
  FROM pagamento
  LEFT JOIN data
  ON pagamento.dataPK = data.dataPK
  LEFT JOIN equipe
  ON pagamento.equipePK = equipe.equipePK
  WHERE dataAno = 2017
  GROUP BY filialRegiaoNome
  ) despesas
```

```
FULL OUTER JOIN (
  SELECT filialRegiaoNome AS REGIAO, ROUND(SUM(receita), 2) AS TOTALRECEITAEQUIPE
  FROM negociacao
  LEFT JOIN data
  ON negociacao.dataPK = data.dataPK
  LEFT JOIN equipe
  ON negociacao.equipePK = equipe.equipePK
  WHERE dataAno = 2017
  GROUP BY filialRegiaoNome
  ) receitas
ON despesas.REGIAO = receitas.REGIAO
ORDER BY TOTALRECEITAEQUIPE DESC
'''

# Guardando a resposta no nome da questão
Questao5 = spark.sql(query)

# Exibindo a resposta da questão na saída padrão com .show()
Questao5.show(25, truncate=False)

## Gerando o arquivo no formato csv com a resposta da questão
Questao5\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("Questao5.csv", index=False, header=True)
```

```
+------------+------------------+------------------+
|REGIAO      |TOTALRECEITAEQUIPE|TOTALDESPESAEQUIPE|
+------------+------------------+------------------+
|SUDESTE     |6524616.11        |7957065.63        |
|CENTRO-OESTE|675807.24         |1818214.21        |
+------------+------------------+------------------+
```

Questão 6 (valor: 1,5)

Liste todas as agregações que podem ser geradas para a partir da soma dos salários por nome do estado da filial e por ano, considerando apenas o trimestre 1 e os funcionários cuja região na qual eles moram é a mesma região na qual a filial está localizada. Arredonde a soma dos salários para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "ESTADO", "ANO", "TOTALRECEITA". Ordene as linhas exibidas primeiro pelo total de receita em ordem descendente, depois por estado em ordem descendente, depois pelo ano em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP usando os métodos de pyspark.sql.

Resposta da Questão 6

```
# Resposta da Questão 6
```

```
# Fazendo a consulta especificada
# Primeiro fazer o join() das 4 tabelas necessárias especificando as junções com on = " "
# Depois usar select() para escolher as colunas de saída
# Usar filter() para filtrar dataTrimestre desejado e onde "filialRegiaoNome = funcRegiaoN
# Usar cube( ) para listar todas as agregações possíveis a partir de "filialEstadoNome", '
# ...e a agregação de "salario" com sum()
# Ordenar em ordem descendente com orderBy e desc
# Arrendondar valores com withColumn() e round()
# Renomear colunas com withColumnRenamed()

Questao6 = pagamento.join(data, on="dataPK") \
  .join(equipe, on="equipePK")\
  .join(funcionario, on="funcPK")\
  .select("filialEstadoNome", "dataAno", "salario")\
  .filter("dataTrimestre = 1 AND filialRegiaoNome = funcRegiaoNome")\
  .cube("filialEstadoNome", "dataAno").sum("salario")\
  .orderBy(desc("sum(salario)"), desc("filialEstadoNome"), desc("dataAno")) \
  .withColumn("sum(salario)", round("sum(salario)",2))\
  .withColumnRenamed("filialEstadoNome", "ESTADO")\
  .withColumnRenamed("dataAno", "ANO")\
  .withColumnRenamed("sum(salario)", "TOTALRECEITA")

# Exibindo a resposta da questão na saída padrão com .show()
Questao6.show(25, truncate=False)

# # Gerando o arquivo no formato csv com a resposta da questão
Questao6\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("Questao6.csv", index=False, header=True)
```

```
+--------------+----+------------+
|ESTADO        |ANO |TOTALRECEITA|
+--------------+----+------------+
|null          |null|9342133.85  |
|SAO PAULO     |null|5657776.45  |
|RIO DE JANEIRO|null|3549567.87  |
|null          |2020|2450982.1   |
|null          |2019|2450982.1   |
|null          |2018|2065272.34  |
|null          |2017|1550525.2   |
|SAO PAULO     |2020|1514279.37  |
|SAO PAULO     |2019|1514279.37  |
|SAO PAULO     |2018|1195964.37  |
|SAO PAULO     |2017|957678.16   |
|RIO DE JANEIRO|2020|869307.97   |
|RIO DE JANEIRO|2019|869307.97   |
|RIO DE JANEIRO|2018|869307.97   |
|null          |2016|824372.11   |
|RIO DE JANEIRO|2017|592847.04   |
|SAO PAULO     |2016|475575.18   |
|RIO DE JANEIRO|2016|348796.92   |
|PERNAMBUCO    |null|134789.52   |
|PERNAMBUCO    |2020|67394.76    |
|PERNAMBUCO    |2019|67394.76    |
+--------------+----+------------+
```

7.3 Análise Relacionada a Totais

O objetivo da análise desta seção é obter uma tabela de totais.

Questão 7 (valor: 1,5)

Liste, para cada nome da região da filial, o número total de funcionários diferentes, o número total de clientes diferentes e o número total de equipes diferentes. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "REGIAO", "TOTALFUNCIONARIOS", "TOTALCLIENTES", "TOTALEQUIPES". Ordene as linhas exibidas primeiro pela região em ordem descendente, depois pelo total de funcionários em ordem descendente, depois pelo total de clientes em ordem descendente, depois pelo total de equipes em ordem descendente. Liste as primeiras 25 linhas da resposta, sem truncamento das *strings*.

Resolva a questão especificando a consulta OLAP na linguagem SQL.

Resposta da Questão 7

```
# Resposta da Questão 7

# Fazendo a consulta especificada
# Primeiro selecionar as colunas de saída com SELECT e renomeá-las com AS...
# ... selecionar os valores distintos das chaves primárias desejadas com DISTINCT()...
# ... e agregar resultados com COUNT()
# Fazer o JOIN das tabelas com as informações requeridas e especificar as junções com ON
# Agrupar filialRegiaoNome com GROUP BY
# Ordenar com ORDER BY e DESC na sequência desejada

query = """
SELECT filialRegiaoNome AS REGIAO, COUNT(DISTINCT(funcPK)) AS TOTALFUNCIONARIOS, COUNT(DIS
FROM equipe JOIN pagamento ON pagamento.equipePK = equipe.equipePK
            JOIN negociacao ON negociacao.equipePK = equipe.equipePK
GROUP BY filialRegiaoNome
ORDER BY REGIAO DESC, TOTALFUNCIONARIOS DESC, TOTALCLIENTES DESC, TOTALEQUIPES DESC
"""

# Guardando a resposta no nome da questão
Questao7 = spark.sql(query)

# Exibindo a resposta da questão na saída padrão com .show()
Questao7.show(25, truncate=False)

## Gerando o arquivo no formato csv com a resposta da questão
Questao7\
  .coalesce(1).limit(25) \
  .toPandas().to_csv("Questao7.csv", index=False, header=True)
```

```
+------------+-----------------+-------------+------------+
|REGIAO      |TOTALFUNCIONARIOS|TOTALCLIENTES|TOTALEQUIPES|
+------------+-----------------+-------------+------------+
|SUDESTE     |134              |138          |7           |
|NORDESTE    |20               |55           |1           |
|CENTRO-OESTE|46               |86           |2           |
+------------+-----------------+-------------+------------+
```

8 Estendendo a Aplicação da BI Solutions

A aplicação da BI Solutions está sendo estendida de forma a analisar um novo assunto de interesse: os gastos realizados na compra de equipamentos. Para tanto, é necessário criar uma nova tabela de dimensão chamada Equipamento, a qual tem como objetivo armazenar dados de equipamentos, os quais devem ser obtidos a partir de 3 fontes de dados heterogêneas.

8.1 Detalhamento das Fontes

Considere que o processo de integração de dados já tenha sido realizado. Como resultado, as 3 fontes de dados (Fonte1, Fonte2, Fonte3) possuem os mesmos atributos, com os mesmos nomes. Esses atributos encontram-se listados a seguir, sendo seus nomes semânticos.

- equipamentoPK
- equipamentoNome
- equipamentoCor
- equipamentoTipo
- equipamentoMoeda
- equipamentoPreco

```
# Obtenção dos dados da Fonte1 e armazenamento desses dados no Dataframe chamado Fonte1
Fonte1 = pd.read_csv('https://raw.githubusercontent.com/CristinaAguiar/QuestaoIntegra2021/
Fonte1.head()
```

	equipamentoPK	equipamentoNome	equipamentoDescricao	equipamentoCor	equipamentc
0	1	monitor	monitor LCD LG 15 polegadas	preto	equip inforn
1	2	monitor	monitor LCD ITAUTEC 19 polegadas	preto	equip inforn
2	3	monitor	monitor LCD AOC 17 polegadas	preto	Equip inforn
3	5	NaN	LENOVO THINKCENTRE MT-M W/ INTEL CORE 2 DUO	preto	equip inforn

```
# Obtenção dos dados da Fonte2 e armazenamento desses dados no Dataframe chamado Fonte2
Fonte2 = pd.read_csv('https://raw.githubusercontent.com/CristinaAguiar/QuestaoIntegra2021/
Fonte2.head()
```

	equipamentoPK	equipamentoNome	equipamentoDescricao	equipamentoCor	equipamentc
0	1	monitor	MONITOR LCD ITAUTEC 17 polegadas	PRETO	equip inforn
1	4	computador	LENOVO THINKCENTRE MT-M W/ INTEL CORE 2 DUO @ ...	PRETO	equip inforn
2	10	computador	HP COMPAQ DC5850 W/ AMD PHENOM X4 4GB RAM 250G...	PRATA	equip inforn
3	11	computador	HP COMPAQ DC5850 W/ AMD PHENOM X4	PRETO	equip inforn

```
# Obtenção dos dados da Fonte3 e armazenamento desses dados no Dataframe chamado Fonte3
Fonte3 = pd.read_csv('https://raw.githubusercontent.com/CristinaAguiar/QuestaoIntegra2021/
Fonte3.head()
```

	equipamentoPK	equipamentoNome	equipamentoDescricao	equipamentoCor	equipamentc
0	1	MONITOR	MONITOR LCD ITAUTEC 17 polegadas	prata	equip inforn
1	5	COMPUTADOR	LENOVO THINKCENTRE MT-M W/ INTEL CORE 2 DUO @ ...	prata	equip inforn
2	6	COMPUTADOR	ITAUTEC INFOWAY SM 3322 W/ AMD PHENOM X2 8GB 3...	prata	equip inforn
3	7	COMPUTADOR	POSITIVO PLUS R70 W/ MOBO ITAUTEC	preto	equip inforn

8.2 Detalhamento da tabela de dimensão Equipamento

A tabela de dimensão Equipamento da BI Solutions deve possuir os seguintes atributos:

- **equipamentoPK**, correspondente aos atributos de mesmo nome nas fontes de dados.
- **equipamentoNome**, correspondente aos atributos de mesmo nome nas fontes de dados.
- **equipamentoDescricao**, correspondente aos atributos de mesmo nome nas fontes de dados.
- **equipamentoCor**, correspondente aos atributos de mesmo nome nas fontes de dados.
- **equipamentoTipo**, correspondente aos atributos de mesmo nome nas fontes de dados.

- **equipamentoPreco**, correspondente aos atributos equipamentoMoeda e equipamentoPreco nas fontes de dados.

8.3 Regras de Negócio do Processo de Integração de Instâncias

No processo de integração de instâncias, devem ser consideradas as seguintes regras de negócio:

- A integração deve ser feita pelo atributo equipamentoPK. Equipamentos que possuam o mesmo valor desse atributo referem-se ao mesmo equipamento.
- Todas as *strings* devem ser escritas em letras maiúsculas, sem acento e sem o uso de caracteres especiais.
- Os valores das *strings* não devem ser truncados.
- Os preços dos equipamentos devem ser armazenados somente em reais. Portanto, para se calcular os valores da coluna equipamentoPreco da tabela de dimensão Equipamento, deve ser feito o cálculo desse valor em reais, utilizando os atributos equipamentoMoeda e equipamentoPreco presentes nas fontes de dados originais. Considere, para isso, as seguintes conversões: (i) 1 dolar USD = 5 reais; e (ii) 1 euro EUR = 6 reais.
- Os preços dos equipamentos devem ter duas casas decimais e não devem incluir a sigla "R$".
- As cores dos equipamentos devem ser armazenadas por meio de números, da seguinte forma:
    - 1: correspondente à cor PRETO nas fontes de dados
    - 2: correspondente à cor AZUL nas fontes de dados
    - 3: correspondente à cor BRANCO nas fontes de dados
    - 4: correspondente à cor PRATA nas fontes de dados
    - 5: correspondente à cor VERMELHO nas fontes de dados
    - 6: correspondente à cor AMARELO nas fontes de dados
- Para resolver inconsistências nos valores de cada atributo que aparecem nas diferentes fontes, desconsidere os valores nulos e considere que:
    - (i) quando em uma coluna o valor for igual nas três fontes, esse valor deve ser armazenado na tabela de dimensão Equipamento na coluna equivalente. Por exemplo, se o nome do equipamento de PK = 1 for caneta nas três fontes de dados, então o valor a ser armazenado é CANETA.
    - (ii) quando em uma coluna o valor for igual em duas fontes e diferente na terceira fonte, o valor a ser armazenado na tabela de dimensão Equipamento na coluna equivalente é o valor que aparece nas duas fontes. Por exemplo, se o nome do

equipamento de PK = 1 for caneta em duas fontes de dados e borracha na terceira fonte de dados, então o valor a ser armazenado é CANETA.

- (iii) quando em uma coluna quando o valor for diferente nas três fontes, escolhe-se por armazenar o valor da Fonte 1 na tabela de dimensão Equipamento na coluna equivalente. Caso o valor da Fonte 1 seja nulo (inexistente), escolhe-se por armazenar o valor da Fonte 2. Caso o valor da Fonte 2 também seja nulo (inexistente), escolhe-se por armazenar o valor da Fonte 3. Isso significa que Fonte1 é mais confiável do que Fonte2, a qual é mais confiável do que Fonte3.

Questão 8 (valor: 2,0)

Realize a geração da tabela de dimensão Equipamento, considerando os detalhamentos dos atributos das seções 8.1 e 8.2 e as regras de negócio do processo de integração de instâncias definido na seção 8.3. A tabela de dimensão Equipamento deve possuir as colunas na ordem e com os nomes especificados a seguir: equipamentoPK, equipamentoNome, equipamentoDescricao, equipamentoCor, equipamentoTipo, equipamentoPreco. Ordene as linhas exibidas pelo atributo equipamentoPK em ordem **ascendente**. Liste **todas** as linhas da resposta, sem truncamento das *strings*.

Resolva a questão usando Pandas. Coloque comentários detalhados explicando a sua resposta.

Resposta da Questão 8

```
# Resposta da Questão 8

# Eliminando os caracteres especiais e convertendo para maiúsculas
import re
Fonte1 = Fonte1.applymap(lambda s: re.sub('[^0-9a-zA-Z]+', ' ', s).upper() if type(s) == s
Fonte2 = Fonte2.applymap(lambda s: re.sub('[^0-9a-zA-Z]+', ' ', s).upper() if type(s) == s
Fonte3 = Fonte3.applymap(lambda s: re.sub('[^0-9a-zA-Z]+', ' ', s).upper() if type(s) == s

# Convertendo os preços para Reais e eliminando os dados desnecessários
cambio = pd.DataFrame([['BRL',1],['USD',5],['EUR',6]], columns=['equipamentoMoeda','txcamb

Fonte1 = Fonte1.merge(cambio, on='equipamentoMoeda', how='left')
Fonte1['equipamentoPreco'] = Fonte1['equipamentoPreco']*Fonte1['txcambio']
Fonte1.drop(['equipamentoMoeda', 'txcambio'], axis=1, inplace=True)

Fonte2 = Fonte2.merge(cambio, on='equipamentoMoeda', how='left')
Fonte2['equipamentoPreco'] = Fonte2['equipamentoPreco']*Fonte2['txcambio']
Fonte2.drop(['equipamentoMoeda', 'txcambio'], axis=1, inplace=True)

Fonte3 = Fonte3.merge(cambio, on='equipamentoMoeda', how='left')
Fonte3['equipamentoPreco'] = (Fonte3['equipamentoPreco']*Fonte3['txcambio'])
```

```
Fonte3.drop(['equipamentoMoeda', 'txcambio'], axis=1, inplace=True)

# Codificando as cores
Fonte1['equipamentoCor'] = Fonte1['equipamentoCor'].map({'PRETO': 1, 'AZUL':2, 'BRANCO': 3
                                                         'PRETA': 1, 'BRANCA': 3, 'VERMELH
Fonte2['equipamentoCor'] = Fonte2['equipamentoCor'].map({'PRETO': 1, 'AZUL':2, 'BRANCO': 3
                                                         'PRETA': 1, 'BRANCA': 3, 'VERMELH
Fonte3['equipamentoCor'] = Fonte3['equipamentoCor'].map({'PRETO': 1, 'AZUL':2, 'BRANCO': 3
                                                         'PRETA': 1, 'BRANCA': 3, 'VERMELH

# Fazendo a união das fontes de dados
merged_df = Fonte1.merge(Fonte2, on = 'equipamentoPK', how='outer').merge(Fonte3, on = 'eq

# Fazendo a escolha de valores para o dataframe final de acordo com a ocorrência nas 3 for
lista_equipamentos = merged_df['equipamentoPK']
indices_colunas = [1,2,3,4,5]

for eqpto in lista_equipamentos:
  for icol in indices_colunas:
    counter_1, counter_2, counter_3 = 1, 1, 1
    valor_1 = merged_df[merged_df['equipamentoPK']==eqpto].iloc[0, icol]
    valor_2 = merged_df[merged_df['equipamentoPK']==eqpto].iloc[0, icol+5]
    valor_3 = merged_df[merged_df['equipamentoPK']==eqpto].iloc[0, icol+10]
    if pd.isna(valor_1):
      counter_1 += -1
    if pd.isna(valor_2):
      counter_2 += -1
    if pd.isna(valor_3):
      counter_3 += -1
    if valor_1 == valor_2:
      counter_1 += 1
      counter_2 += 1
    if valor_1 == valor_3:
      counter_1 += 1
      counter_3 += 1
    if valor_2 == valor_3:
      counter_2 += 1
      counter_3 += 1
    values_list = [valor_1, valor_2, valor_3]
    counter_list = [counter_1, counter_2, counter_3]
    max_value = max(counter_list)
    max_index = counter_list.index(max_value)
    column_name = merged_df.columns[icol+10]
    merged_df.loc[merged_df['equipamentoPK'] == eqpto, [column_name]] = values_list[max_ir

# Eliminando as colunas adicionais geradas na união das fontes
merged_df.drop(merged_df.columns[[1,2,3,4,5,6,7,8,9,10]], axis=1, inplace=True)

# Convertendo o tipo de dado da Cor de float para int
merged_df['equipamentoCor'] = merged_df['equipamentoCor'].astype(int)

# Ordenando pela coluna 'equipamentoPK'
Questao8 = merged_df.sort_values(by=['equipamentoPK'], ascending=[True])

# Reset do index
```

```
Questao8.reset_index(drop=True, inplace=True)

# arredondando as casas decimais e finalizando a consulta mostrando todos os resultados
Questao8 = Questao8.round(2)

# exibindo a resposta na saída padrão
pd.options.display.max_colwidth = 100
display(Questao8)

# gerando o arquivo no formato csv com a resposta da questão
Questao8.to_csv("Questao8.csv", index=False, header=True)
```

	equipamentoPK	equipamentoNome	equipamentoDescricao	equipamentoCor	equipa
0	1	MONITOR	MONITOR LCD ITAUTEC 17 POLEGADAS	1	INF
1	2	MONITOR	MONITOR LCD ITAUTEC 19 POLEGADAS	1	INF
2	3	MONITOR	MONITOR LCD AOC 17 POLEGADAS	1	INF
3	4	COMPUTADOR	LENOVO THINKCENTRE MT M W INTEL CORE 2 DUO 2 33 GHZ 4GB RAM 320GB HD	1	INF
4	5	COMPUTADOR	LENOVO THINKCENTRE MT M W INTEL CORE 2 DUO 2 33 GHZ 2GB RAM 160GB HD	1	INF
5	6	COMPUTADOR	ITAUTEC INFOWAY SM 3322 W AMD PHENOM X2 8GB 320GB HD	4	INF
6	7	COMPUTADOR	POSITIVO PLUS R70 W MOBO ITAUTEC SM3322 AMD PHENOM X2 4GB 320GB HD	1	INF
7	8	COMPUTADOR	ITAUTEC INFOWAY ST 1430 W MB ITAUTEC SM3322 AMD PHENON X2 6GB 320GB HD	4	INF
8	9	COMPUTADOR	HP COMPAQ DC5850 W AMD PHENOM X4 4GB RAM 250GB HD	4	INF
9	10	COMPUTADOR	HP COMPAQ DC5850 W AMD PHENOM X4 4GB RAM 250GB HD	4	INF
10	11	COMPUTADOR	HP COMPAQ DC5850 W AMD PHENOM X4 8GB RAM 250GB HD	1	INF
11	12	COMPUTADOR	HP COMPAQ DC5850 W AMD PHENOM X4 16GB RAM 250GB HD	4	INF
12	13	COMPUTADOR	HP COMPAQ DC5850 W AMD PHENOM X4 16GB RAM 250GB HD	1	INF
13	14	COMPUTADOR	PAUTA CONNECT BB8701A W MOBO ITAUTEC SM3322 AMD	1	INF

			PHENOM X2 4GB 4HDS 320GB	
14	15	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO COM RODAS REVESTIDA EM TECIDO NA COR AZUL	2
15	16	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO COM RODAS REVESTIDA EM TECIDO NA COR AZUL	2
16	17	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO COM RODAS REVESTIDA EM TECIDO NA COR PRETA	1
17	18	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO COM RODAS REVESTIDA EM TECIDO NA COR AZUL	2
18	19	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO COM RODAS REVESTIDA EM TECIDO NA COR AZUL	2
19	20	CADEIRA DE ESCRITORIO	CADEIRA DE ESCRITORIO REVESTIDA EM TECIDO NA COR AZUL	2
20	21	CADEIRA DE ESCRITORIO	CADEIRA UNIVERSITARIA REVESTIDA EM TECIDO NA COR AZUL COM BRACO	2
21	22	CADEIRA DE ESCRITORIO	CADEIRA UNIVERSITARIA REVESTIDA EM TECIDO NA COR AZUL	2